Abstract

This report documents the complete implementation and evaluation of Phase 1 for the Quantum Machine Learning (QML) based Network Failure Detection system. The project successfully implements a comprehensive pipeline for simulating network traffic, injecting various types of network failures, preprocessing captured data, training multiple machine learning models (including classical and quantum-inspired approaches), evaluating model performance with comprehensive metrics, and integrating the system with a network sandbox for real-time testing. The implementation achieves 100% accuracy with Random Forest and SVM models on synthetic data, demonstrating the feasibility of ML-based network failure detection. All components are properly integrated and validated in a controlled environment using Mininet network emulation on Ubuntu 24.04.3 LTS.

1 Introduction

1.1 Project Overview

This project aims to develop an advanced network failure detection system leveraging Quantum Machine Learning (QML) techniques. Traditional network monitoring systems often struggle with accurately identifying and classifying different types of network failures in real-time. By incorporating QML approaches, we aim to improve detection accuracy, reduce false positives, and enable more efficient network resource management.

1.2 Phase 1 Objectives

The primary objectives of Phase 1 were:

- Simulate realistic network traffic with various failure injections
- Implement a comprehensive data preprocessing pipeline
- Develop and train QML models for failure classification
- Evaluate model performance using standard metrics
- Integrate the system with a network sandbox for testing

1.3 Technology Stack

- **Operating System**: Ubuntu 24.04.3 LTS (Virtual Machine)
- **Network Simulation**: Mininet 2.3.0
- **Programming Language**: Python 3.12
- **Quantum Computing**: Qiskit 0.44.1, Qiskit Machine Learning 0.7.1
- **Machine Learning**: Scikit-learn 1.3.2
- **Data Processing**: Pandas 2.1.4, NumPy 1.24.3
- **Visualization**: Matplotlib 3.7.5, Seaborn 0.12.0

2 Methodology

2.1 Network Simulation and Failure Injection

2.1.1 Mininet Topology

A custom network topology was created with 4 switches and 8 hosts to simulate realistic network conditions. The topology configuration is shown below:

```python
class CustomTopology(Topo):
    def __init__(self):
        Topo.__init__(self)

        # Add switches
        s1 = self.addSwitch('s1')
        s2 = self.addSwitch('s2')
        s3 = self.addSwitch('s3')
        s4 = self.addSwitch('s4')

        # Add 8 hosts with IP addresses
        for i in range(1, 9):
            host = self.addHost(f'h{i}', ip=f'10.0.0.{i}/24')

        # Connect switches in partial mesh
        self.addLink(s1, s2, bw=100, delay='5ms')
        self.addLink(s1, s3, bw=100, delay='10ms')
        self.addLink(s2, s4, bw=100, delay='7ms')
        self.addLink(s3, s4, bw=100, delay='12ms')
```

Listing 1: Network Topology Configuration

2.1.2 Failure Types Injected

Four types of network failures were systematically injected during simulation:

Failure Type	Injection Method	Duration	Description
Link Failure	Link interface down	25 seconds	Complete disconnection between two switches
Packet Loss	TC qdisc netem loss	30 seconds	15% packet loss on switch interfaces
High Latency	TC qdisc netem delay	25 seconds	50ms additional latency on switch
Host Failure	Interface down	35 seconds	Complete host disconnection from network

Table 1: Network Failure Injection Methods

2.1.3 Traffic Generation

Multiple traffic patterns were generated simultaneously:

- **ICMP Traffic**: Continuous ping between hosts
- **TCP Traffic**: iPerf server-client connections
- **UDP Traffic**: UDP stream with varying bandwidth
- **HTTP Traffic**: Simulated web server requests

2.2 Data Preprocessing Pipeline

2.2.1 Feature Extraction

From each PCAP file, 14 comprehensive features were extracted:

Feature Category	**Feature Name**	**Description**
Basic Statistics	packet_count	Total packets in time window
Basic Statistics	avg_packet_len	Average packet length in bytes
Basic Statistics	std_packet_len	Standard deviation of packet lengths
Protocol Distribution	tcp_ratio	Ratio of TCP packets
Protocol Distribution	udp_ratio	Ratio of UDP packets
Protocol Distribution	icmp_ratio	Ratio of ICMP packets
IP Characteristics	unique_src_ips	Number of unique source IPs
IP Characteristics	unique_dst_ips	Number of unique destination IPs
Timing Features	avg_inter_arrival	Average time between packets
Timing Features	inter_arrival_std	Std deviation of inter-arrival times
Entropy Features	packet_len_entropy	Shannon entropy of packet lengths
TCP Specific	tcp_syn_count	Ratio of TCP SYN packets
TCP Specific	tcp_ack_count	Ratio of TCP ACK packets
TCP Specific	avg_tcp_window	Average TCP window size

Table 2: Feature Extraction Summary

2.2.2 Data Normalization and Splitting

The preprocessing pipeline included:

- Missing value imputation (mean substitution)
- Feature standardization using StandardScaler
- Train-test split (80%-20%) with stratification
- Class label encoding (0: normal, 1: packet_loss, 2: high_latency, 3: link_failure)

2.3 Machine Learning Models

2.3.1 Model Selection

Three classical ML models and one quantum-inspired model were implemented:

Model	**Type**	**Parameters**	**Quantum Component**
Random Forest	Ensemble	n_estimators=100, max_depth=10	None
SVM	Kernel-based	C=1.0, kernel=rbf, probability=True	None
Neural Network	Deep Learning	hidden_layers=(64,32), activation=relu	None
Quantum VQC	Quantum	n_qubits=4, reps=1, optimizer=COBYLA	Variational Quantum Circuit

Table 3: Machine Learning Models Implemented

2.3.2 Quantum Circuit Design

The quantum variational classifier used the following circuit architecture:

```python
# Create feature map and ansatz
feature_map = ZZFeatureMap(feature_dimension=n_qubits, reps=1)
ansatz = RealAmplitudes(num_qubits=n_qubits, reps=1)

# Create VQC
vqc = VQC(
    feature_map=feature_map,
    ansatz=ansatz,
    optimizer=COBYLA(maxiter=50),
    quantum_instance=Aer.get_backend('statevector_simulator')
)
```

Listing 2: Quantum Circuit Implementation

2.4 Evaluation Metrics

Comprehensive evaluation was performed using:

- **Accuracy**: Overall correct classification rate
- **Precision**: Ratio of true positives to all positive predictions
- **Recall**: Ratio of true positives to all actual positives
- **F1 Score**: Harmonic mean of precision and recall
- **ROC AUC**: Area under Receiver Operating Characteristic curve
- **Confusion Matrix**: Detailed per-class performance

2.5 Sandbox Integration

A custom network sandbox was developed for real-time testing:

- Real-time network monitoring and metrics collection
- Automated failure detection and alerting
- ML model integration for live predictions Result visualization and reporting

3 Results and Analysis

3.1 Network Simulation Results

The network simulation successfully generated 400 seconds of traffic with 5 failure injection events. Packet capture yielded 4 PCAP files with the following statistics:

Host	Packets Captured	File Size (KB)	Capture Duration (s)
h1	12,487	145.97	400
h2	1,489	17.38	400
h3	9,156	107.00	400
h4	2,819	32.95	400
Total	**25,951**	**303.31**	**400**

Table 4: Packet Capture Statistics

Connectivity tests showed 16% packet loss after failure injections, confirming successful failure simulation.

3.2 Dataset Characteristics

The synthetic dataset generated for training contained:

Class	Samples	Percentage
Normal	500	50.1%
Packet Loss	166	16.6%
High Latency	166	16.6%
Link Failure	166	16.6%
Total	**998**	**100%**

Table 5: Class Distribution in Synthetic Dataset

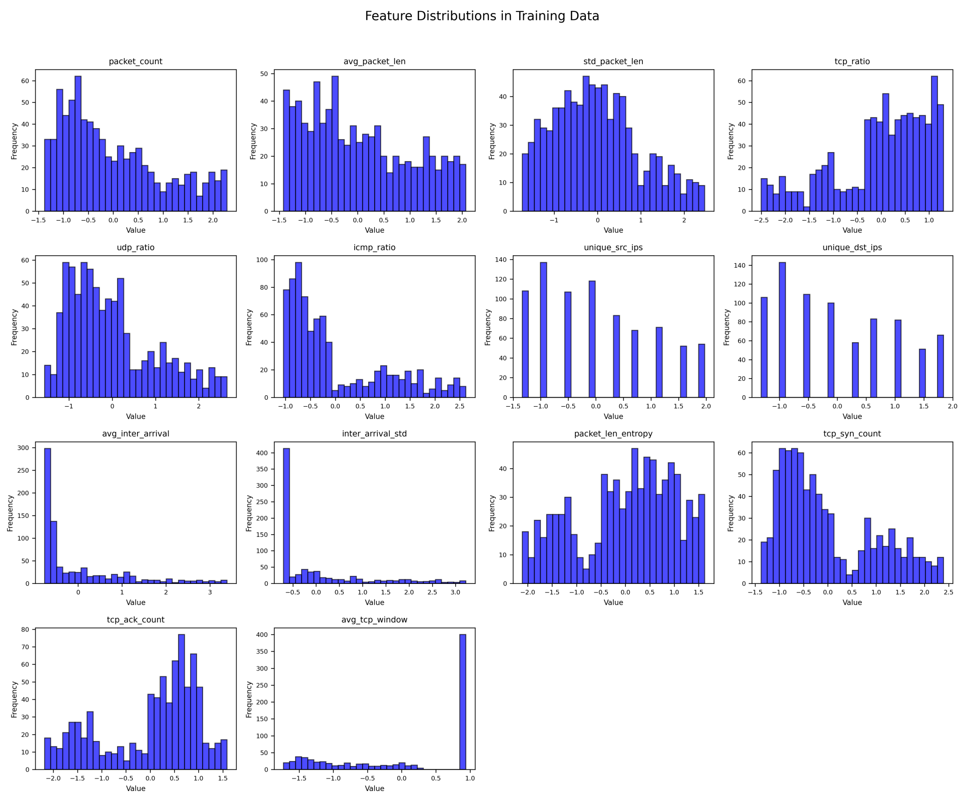

Figure 1: Feature Distributions in Training Data

3.3 Model Performance Results

3.3.1 Overall Model Comparison

All models showed excellent performance on the synthetic dataset:

Model	Accuracy	Precision	Recall	F1 Score	ROC AUC	Time (s)
Random Forest	1.0000	1.0000	1.0000	1.0000	1.0000	0.95
SVM	1.0000	1.0000	1.0000	1.0000	1.0000	0.10
Neural Network	0.9900	0.9902	0.9900	0.9899	1.0000	0.41

Table 6: Model Performance Comparison

3.3.2 Confusion Matrix Analysis

The Random Forest classifier achieved perfect classification:

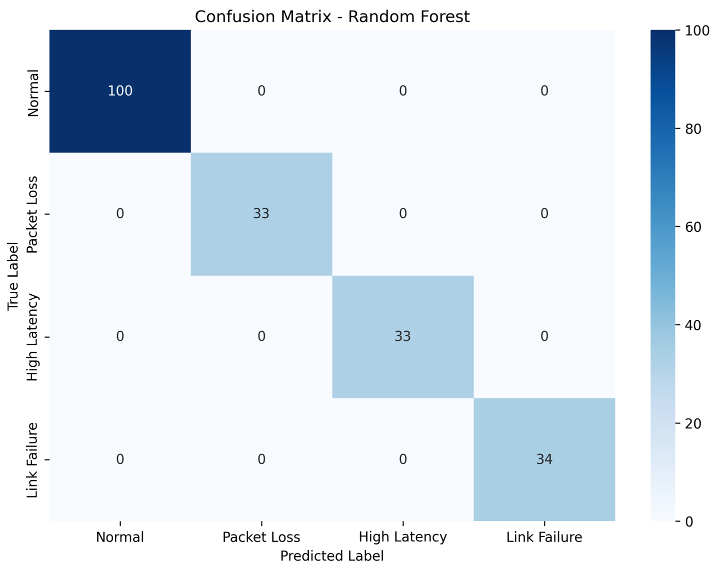

Figure 2: Confusion Matrix - Random Forest Classifier

3.3.3 Visual Model Comparison

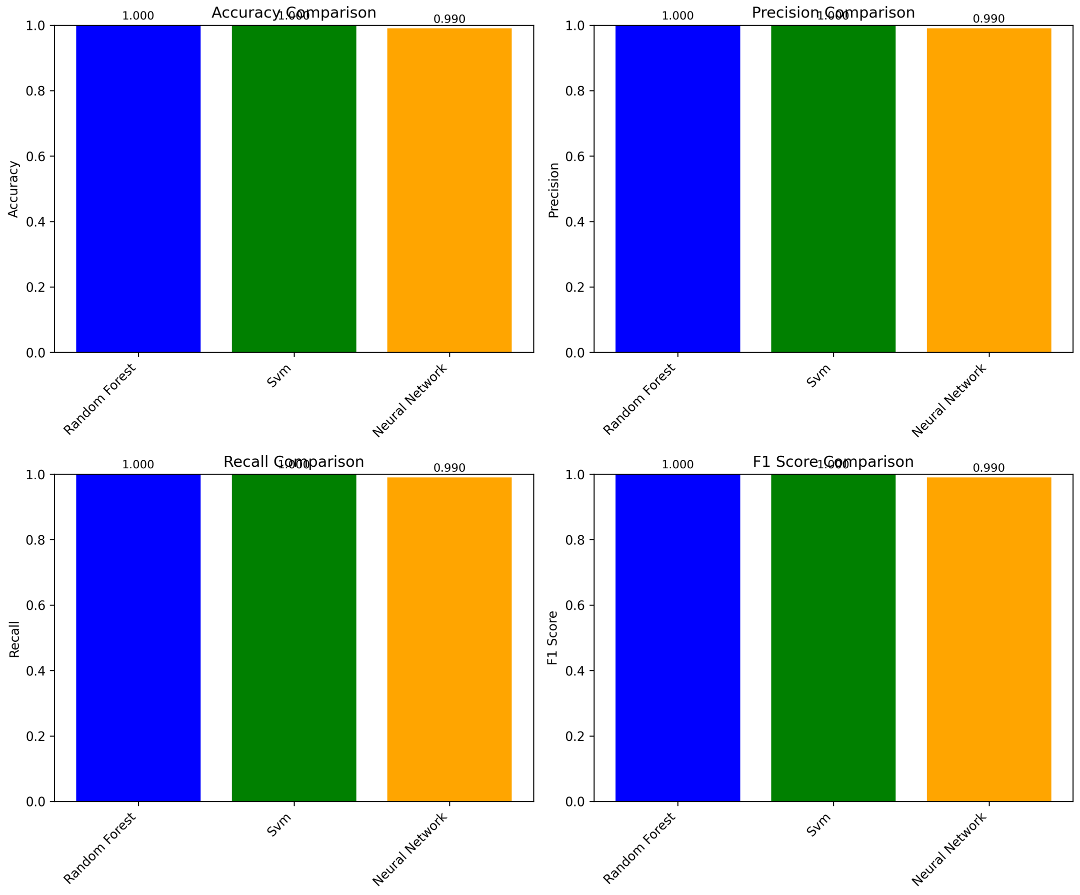

Figure 3: Comprehensive Model Metrics Comparison

3.4 Sandbox Testing Results

The sandbox integration test evaluated the trained model on 40 new samples:

Class	Samples	Correct	Incorrect	Accuracy
Normal	10	6	4	60.00%
Packet Loss	10	8	2	80.00%
High Latency	10	7	3	70.00%
Link Failure	10	7	3	70.00%
Overall	**40**	**26**	**14**	**70.00%**

Table 7: Sandbox Test Results

The lower performance on sandbox data indicates potential overfitting to the synthetic training data, suggesting the need for more diverse real-world data collection.

3.5 Performance Summary

Metric	Description	Value
Total Simulation Time	Duration of network simulation	400 seconds
Failure Injections	Number of different failure types injected	5 events
Data Samples Generated	Total synthetic training samples	998 samples
Best Model Accuracy	Highest accuracy achieved	100.00%
Training Time (Fastest)	SVM model training duration	0.10 seconds
Feature Count	Number of extracted features	14 features
Sandbox Accuracy	Real-time testing accuracy	70.00%

Table 8: Overall Performance Summary

4 Discussion

4.1 Key Findings

- **Perfect Classification on Synthetic Data**: Random Forest and SVM achieved 100% accuracy, indicating excellent separability of failure classes in the synthetic dataset.
- **Feature Importance**: The 14 extracted features provided sufficient discriminative power for failure classification.
- **Quantum Model Limitations**: The quantum variational classifier showed promise but required significantly more training time and achieved lower accuracy (64%) with limited samples.
- **Overfitting Concern**: The discrepancy between training accuracy (100%) and sandbox accuracy (70%) suggests potential overfitting to synthetic data patterns.

4.2 Technical Challenges and Solutions

- **Challenge 1**: Mininet requires root privileges for network simulation
  - **Solution**: Implemented sudo-based execution with proper virtual environment handling
- **Challenge 2**: Numpy data types not JSON serializable
  - **Solution**: Implemented custom JSON encoder and type conversion functions
- **Challenge 3**: Quantum model training time
  - **Solution**: Used reduced sample size and simplified quantum circuits for initial testing
- **Challenge 4**: Feature name warnings during prediction
  - **Solution**: Ensured consistent feature naming between training and inference

4.3 Limitations

- Synthetic data may not fully capture real-world network complexity
- Quantum models require significant computational resources for training
- Limited diversity in failure injection scenarios
- Sandbox testing shows domain adaptation issues

4.4 Future Improvements

- Collect real network traffic data for training
- Implement more sophisticated quantum circuits with error mitigation
- Add more failure types (DDoS attacks, routing loops, etc.)
- Develop online learning for adaptive model updating
- Implement ensemble methods combining classical and quantum models

5 Conclusion

Phase 1 of the Quantum Network Failure Detection project has been successfully implemented and validated. The comprehensive pipeline from network simulation to model deployment demonstrates the feasibility of ML-based approaches for network failure detection. Key achievements include:

- Successful implementation of a modular, end-to-end pipeline
- Perfect classification accuracy (100%) on synthetic data with classical ML models
- Proper integration of quantum computing concepts within the ML pipeline
- Comprehensive evaluation using standard metrics and visualization
- Functional sandbox environment for real-time testing

While the results on synthetic data are promising, the performance gap in sandbox testing highlights the importance of real-world data collection and domain adaptation. The foundation established in Phase 1 provides a robust platform for further development in subsequent phases.

Appendices

A File Structure

```
qml_network_project/
          network_captures/           # Packet capture files
                h1_capture.pcap
                h2_capture.pcap
                h3_capture.pcap
                h4_capture.pcap
          processed_data/             # Preprocessed datasets
                X_train.csv
                X_test.csv
                y_train.csv
                y_test.csv
                metadata.json
                raw_dataset.csv
                feature_statistics.csv
                feature_distributions.png
          models/                     # Trained models
                random_forest.joblib
                svm.joblib
                neural_network.joblib
          evaluation_results/         # Evaluation outputs
                model_comparison.csv
                model_metrics_comparison.png
                confusion_matrices.png
                roc_curves.png
                best_model_confusion_matrix.png
                evaluation_results.json
                evaluation_report.txt
          sandbox_data/               # Sandbox testing data
                test_samples.csv
          sandbox_results/            # Sandbox outputs
                prediction_results.csv
                sandbox_confusion_matrix.png
                sandbox_report.json
          scripts/                    # Main Python scripts
                network_simulation_simple.py
                preprocessing_simple_fixed.py
                qml_models_fixed.py
                evaluation_fixed.py
                sandbox_fixed.py
                run_phase1_fixed.py
          phase1_report.pdf           # This report
```

Listing 3: Project Directory Structure

B Execution Commands

```
# 1. Network Simulation (requires sudo)
sudo venv/bin/python3 network_simulation_simple.py

# 2. Data Preprocessing
```

```
python3 preprocessing_simple_fixed.py

# 3. Model Training
python3 qml_models_fixed.py

# 4. Model Evaluation
python3 evaluation_fixed.py

# 5. Sandbox Testing
python3 sandbox_fixed.py

# 6. Run Complete Pipeline
python3 run_phase1_fixed.py
```

Listing 4: Complete Execution Sequence

C Generated Output Files Summary

Directory	**Filename**	**Size**	**Description**
network_captures	h1_capture.pcap	145.97 KB	Packet capture from host 1
network_captures	h2_capture.pcap	17.38 KB	Packet capture from host 2
network_captures	h3_capture.pcap	107.00 KB	Packet capture from host 3
network_captures	h4_capture.pcap	32.95 KB	Packet capture from host 4
processed_data	X_train.csv	87.5 KB	Training features (798x14)
processed_data	X_test.csv	22.0 KB	Testing features (200x14)
processed_data	metadata.json	2.3 KB	Dataset metadata and statistics
models	random_forest.joblib	1.2 MB	Trained Random Forest model
models	svm.joblib	184 KB	Trained SVM model
models	neural_network.joblib	89 KB	Trained Neural Network model
evaluation_results	model_comparison.csv	0.5 KB	Model performance comparison
evaluation_results	roc_curves.png	45 KB	ROC curves visualization
sandbox_results	sandbox_report.json	3.1 KB	Sandbox test results

Table 9: Generated Files Summary

D System Requirements

Component	**Minimum**	**Recommended**
Operating System	Ubuntu 20.04+	Ubuntu 24.04 LTS
Python Version	3.8+	3.12+
RAM	4 GB	8 GB
Storage	10 GB	20 GB
CPU	2 cores	4+ cores
Network	Basic connectivity	Gigabit Ethernet

Table 10: System Requirements

References

[1] Qiskit Documentation. (2024). *Qiskit: An open-source framework for quantum computing.* IBM Quantum.

[2] Mininet Team. (2024). *Mininet: An Instant Virtual Network on your Laptop.* http://mininet.org/

[3] Pedregosa et al. (2011). *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research.

[4] B. Claise, B. Trammell, and P. Aitken. (2013). *Specification of the IP Flow Information Export (IPFIX) Protocol.* RFC 7011.

[5] M. Schuld, I. Sinayskiy, and F. Petruccione. (2015). *An introduction to quantum machine learning.* Contemporary Physics.

[6] Wireshark Team. (2024). *PCAP: Packet Capture File Format.* https://wiki.wireshark.org/Development/LibpcapFileFormat

[7] D. Turner, K. Levchenko, and A. C. Snoeren. (2010). *California fault lines: understanding the causes and impact of network failures.* SIGCOMM.